

ตารางขนาด 8×8
 $2^n \times 2^n$ เมื่อ $n \geq 0$

$n = 1$

1 2 3 4 ?

5

count

1 1 2 3 5 ?



1 2 3 4 6 9 ?

1 2 6 24 ?

Recurrence Relation and Recursive Function

ความสัมพันธ์เวียนเกิด
และฟังก์ชันเรียกตัวเอง

ความสัมพันธ์เวียนเกิด (Recurrence Relation)

- ความสัมพันธ์เวียนเกิดสำหรับลำดับ $\{a_n\}$ คือ สมการที่แสดงความสัมพันธ์ระหว่างพจน์ a_n กับพจน์ก่อนหน้า ซึ่งอาจเกิดจากพจน์ก่อนหน้ามากกว่า 1 พจน์

$$a_n, a_{n-1}, a_{n-2}, \dots, a_1, a_0$$

เมื่อ $n \geq n_0$ และ n_0 เป็นจำนวนเต็มที่มากกว่าหรือเท่ากับ 0

ตัวอย่างเช่น

$$a_n = a_{n-1} + 1 \quad \text{*กรณีความสัมพันธ์ที่เกิดจากพจน์ก่อนหน้าเพียงหนึ่งพจน์}$$

$$a_n = a_{n-1} + a_{n-2} \quad \text{*กรณีความสัมพันธ์ที่เกิดจากพจน์ก่อนหน้ามากกว่าหนึ่งพจน์}$$

ความสัมพันธ์เวียนเกิด (Recurrence Relation)

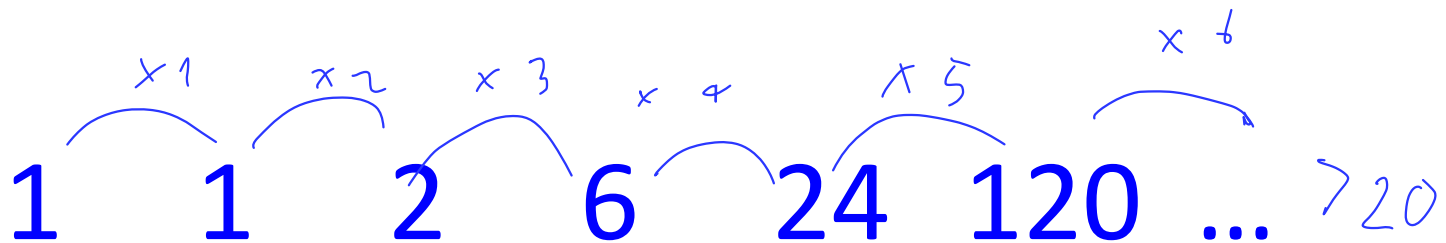
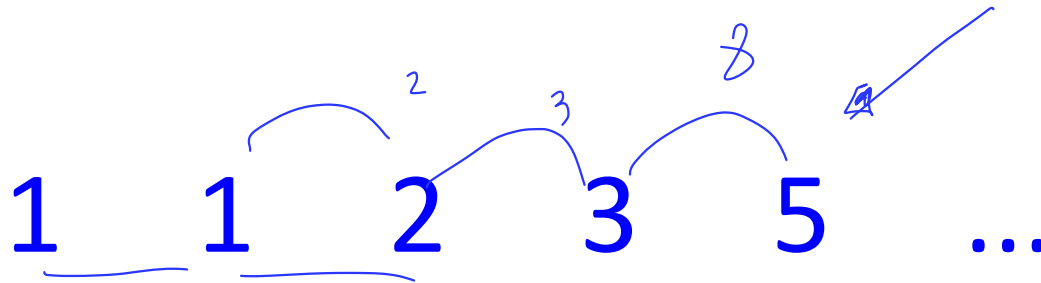
ลำดับตัวเลข คือ 2 4 6 8 ...

ความสัมพันธ์เวียนเกิด คือ อะไร

ลำดับตัวเลข คือ 1 3 6 10 15 ...

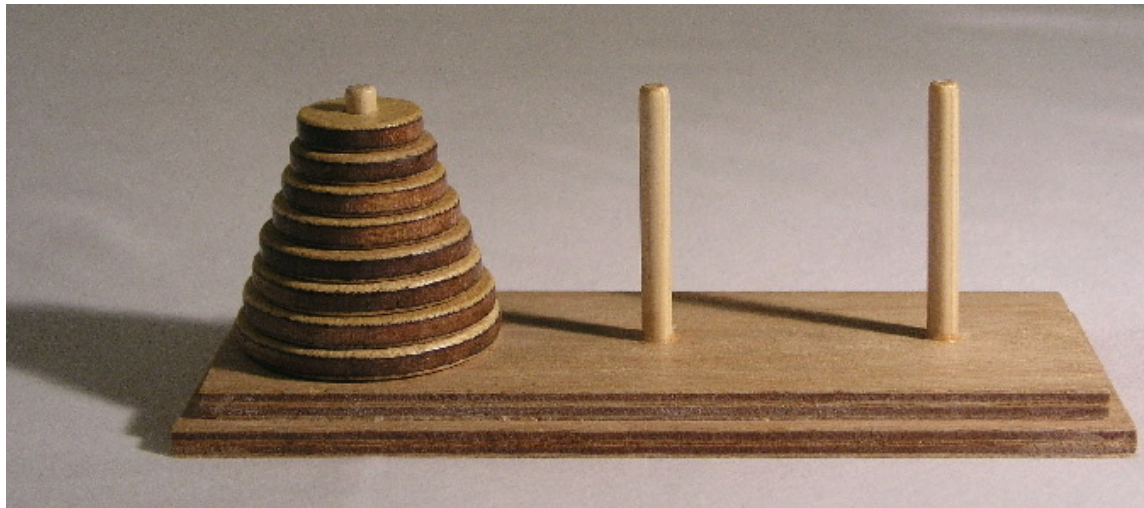
ความสัมพันธ์เวียนเกิด คือ อะไร

ความสัมพันธ์เวียนเกิด (Recurrence Relation)



Tower of Hanoi

- A classic problem of recurrence relation and recursive function.
- หาจำนวนครั้งของการย้ายจาน (disc) ทั้งหมด (n จาน) จากหลักหนึ่งไปยังอีกหลักหนึ่ง โดยมีเงื่อนไขว่า ย้ายได้ครั้งละหนึ่งจาน ย้ายไปซ้อนทับกันได้แต่จานเล็กต้องอยู่บนจานที่ใหญ่กว่าเสมอ





Move $N-1$ smallest discs to pole B.



Move largest disc to pole C.



Move $N-1$ smallest discs to pole C.

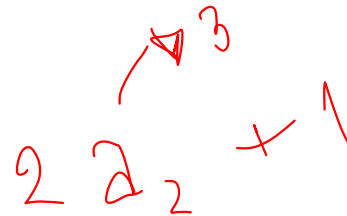


Hanoi Tower Prob.

- จำนวนครั้งของการสลับ n แผ่น

$$= a_{n-1} + 1 + a_{n-1}$$

$$= 2a_{n-1} + 1$$



หรือ คำตอบเฉพาะ คือ

$$= 2^n - 1$$

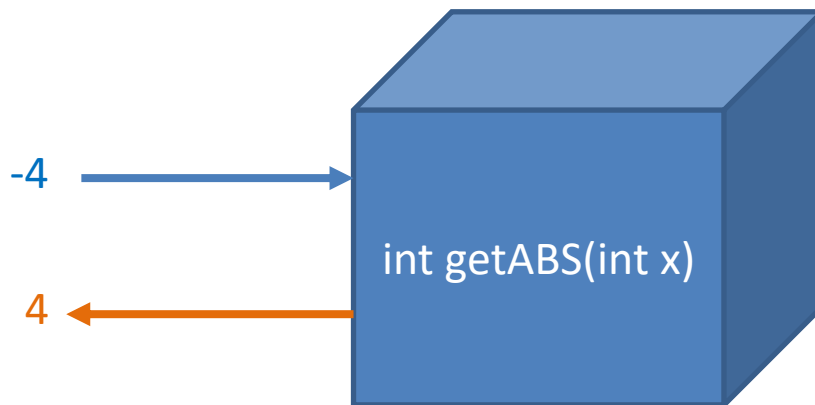
Function

- ฟังก์ชัน คือ ส่วนย่อยของโปรแกรม ที่สร้างขึ้นเพื่อวัตถุประสงค์การทำงานอย่างใดอย่างหนึ่ง เพื่อความซ้ำซ้อนของโปรแกรม จัดโครงสร้างโปรแกรมให้เป็นระบบและง่ายต่อการตรวจสอบแก้ไข
- ถ้าเปรียบไฟล์โปรแกรม 1 โปรแกรมเหมือนบ้านหนึ่งหลัง ห้องต่างๆ ภายในบ้าน เช่น ห้องรับแขก ห้องน้ำ ห้องครัว ก็เป็นเสมือนองค์ประกอบย่อยภายในโปรแกรมที่ถูกแบ่งตามฟังก์ชันงานของโปรแกรมนั้น แต่ห้องเราใช้ซ้ำได้ ฟังก์ชันเหล่านั้นก็สามารถถูกเรียกใช้ซ้ำกี่ครั้งก็ได้เช่นกัน

Function

- ฟังก์ชัน

```
int getABS(int x) {  
    if(x<0)  
        return -x;  
    else return x;  
}  
  
int main() {  
    int rs = getABS(-4);  
    printf("%d",rs);  
}
```



- 4 คือ ค่าพารามิเตอร์ที่ส่งไปยังฟังก์ชัน
(ทำให้พารามิเตอร์ x มีค่าเท่ากับ -4)
- 4 คือ ค่าผลลัพธ์ที่ถูกส่งกลับมา

Recursive Function

- ฟังก์ชันเรียกตัวเอง
- องค์ประกอบ
 - Recursive form (ได้จาก recurrence relation)
 - Stop condition (ได้จากพจน์เริ่มต้นของ recurrence relation)
- ตัวอย่าง
 - จำนวนของแบคทีเรีย ณ ชั่วโมงที่ n เมื่อมีแบคทีเรียเริ่มต้นเท่ากับ 5
$$a_n = 2a_{n-1} \text{ เมื่อ } a_0 = 5 \text{ (พจน์เริ่มต้น)}$$

Recursive Function

2

1 52
2x

ตัวอย่าง

การคำนวณหาจำนวนของแบคทีเรีย ณ ชั่วโมงที่ n เมื่อมี
แบคทีเรียเริ่มต้นเท่ากับ 5

$$a_n = 2a_{n-1} \quad \text{เมื่อ } a_0 = 5$$

จากความสัมพันธ์เวียนเกิด

$$a_n = 2a_{n-1} \quad \text{คือ recursive form}$$

$$n = 0 \quad \text{คือ stop condition ซึ่งให้ค่า } a_0 = 5$$

Recursive Function

$a_n = 2a_{n-1}$ คือ recursive form

$n = 0$ คือ stop condition ซึ่งให้ค่า $a_0 = 5$

//สมมติ numb (n) แทน a_n หรือจำนวนแบคที่เรียชั่วโม่งที่ n

```
int numb (int n) {  
    if (n==0)        //stop condition  
        return 5;    //ถ้า n=0, a(n)=a(0)=5  
    else  
        return (2*numb(n-1)); //a(n)=2*a(n-1) ถ้า n>0  
}
```

ตัวอย่างหาจำนวนแบคทีเรีย ณ ชั่วโมงที่ 5 --> numb(5)

$$\begin{aligned}\text{numb}(5) &= 2 * \text{numb}(4) \\ &= 2 * 2 * \text{numb}(3) \\ &= 2 * 2 * 2 * \text{numb}(2) \\ &= 2 * 2 * 2 * 2 * \text{numb}(1) \\ &= 2 * 2 * 2 * 2 * 2 * \text{numb}(0)\end{aligned}$$

1 2 3 4 5

recursive call
(forward)

Stop calling when n=0

$$\begin{aligned}&= 2 * 2 * 2 * 2 * 2 * 5 \\ &= 2 * 2 * 2 * 2 * 10 \\ &= 2 * 2 * 2 * 20 \\ &= 2 * 2 * 40 \\ &= 2 * 80 \\ &= 160\end{aligned}$$

return the result
(backward,
automatically)

Recursive Function

โจทย์ หาผลรวมของลำดับตัวเลขตั้งแต่ 1 ถึง n ($1+2+3+4+5+..+n$)

$a_n = n + a_{n-1}$ คือ recursive form เมื่อ a_n คือผลรวมจนถึงพจน์ที่ n

$n = 1$ คือ stop condition ซึ่งให้ค่า $a_1 = 1$
(เมื่อ $n \geq 1$)

//สมมติ $\text{sum}(n)$ แทน a_n หรือ ผลรวมของลำดับตัวเลขจนถึงพจน์ที่ n

```
int sum(int n) {  
    if (n==1)    //stop condition  
        return 1;    //a1 = 1 ถ้า n=1  
    else    //return recursive form  
        return (n+sum(n-1));    //an = n + an-1 ถ้า n>1  
}
```

 Recursive call

Recursive Function

ตัวอย่างที่ 4.1 การหาผลบวกของเลขจำนวนเต็มตั้งแต่ 1 ถึง n มีความสัมพันธ์ดังนี้

Recursive form คือ $\text{sum}(n) = n + \text{sum}(n-1)$ เมื่อ $n > 1$

เงื่อนไขหยุด (stop condition) คือ $n==1$ ซึ่งให้ค่า $\text{sum}(n) = \text{sum}(1) = 1$

```
int sum(int n) {           //เมื่อ  $n \geq 1$  (pre-condition, assumption)
    if (n==1)              //กรณี  $n=1$  (stop condition)
        return(n);
    else                   //กรณี  $n > 1$ 
        return(n + sum(n-1)); //recursive call
}
```

ตัวอย่างที่ 4.2 การคำนวณหาค่า factorial ด้วยฟังก์ชันเรียกตัวเอง

Ex $5! = 5 \times 4!$

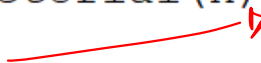
$n! = n \times (n-1)!$

$n! = n * (n-1) * (n-2) * \dots * (1)$ หรือ $n! = n * (n-1)!$ เมื่อ $0! = 1! = 1$ และ $n \geq 0$

เขียนอยู่ในรูป recursive form ได้เป็น $\text{factorial}(n) = n * \text{factorial}(n-1)$

เมื่อ $\text{factorial}(n)$ แทน $n!$

เงื่อนไขหยุด (stop condition) คือ ~~$n=1$~~ ซึ่งให้ค่า $\text{factorial}(n) = 1$

$n == 0$ หรือ $n == 1$ 

```
#include <stdio.h>
long int factorial (int n);
int main () {
    int n;
    printf ("n = "); scanf ("%d", &n);
    printf ("n! = %ld", factorial(n));
}
long int factorial(int n) {
    if ( $n \leq 1$ ) /*stop condition*/  $\text{if}(n == 0 || n == 1)$ 
        return (1);
    else
        return (n*factorial(n-1)); /* recursive form */
}
```

ตัวอย่างที่ 4.4 โปรแกรมคำนวณหาเลขยกกำลัง x^n เมื่อ n มีค่ามากกว่าหรือเท่ากับ 0

สมมติให้ $\text{pow}(x, n)$ แทนฟังก์ชันสำหรับคำนวณหา x^n เมื่อกำหนดให้ $n \geq 0$ จะได้

รูปแบบของการเรียกตัวเอง คือ $\text{pow}(x, n) = x * \text{pow}(x, n-1)$

เงื่อนไขหยุด คือ $n=0$ ซึ่งให้ค่า $\text{pow}(x, n) = \text{pow}(x, 0) = 1$

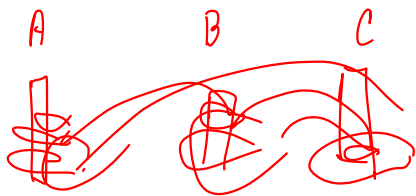
ฟังก์ชันเรียกตัวเองที่สมบูรณ์แสดงได้ดังนี้

```
#include <stdio.h>
void double pow(int x, int n);
int main() {
    printf("Enter x and n:");
    scanf("%d %d", &x, &n);
    printf("Enter pow(x,n)=%0.2f", pow(x, n));
}
double pow(int x, int n) {
    if(n==0) //stop condition, when n==0
        return 1; //pow(x,0)=1
    else //case n>0
        return (x*pow(x,n-1)); //call recursive form
}
```

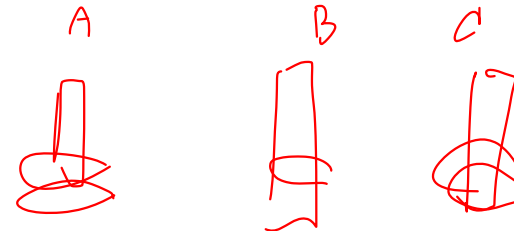
แบบฝึกหัด

$$a_n = a_{n-1} + a_{n-2}$$

1. จงเขียน recursive function เพื่อคำนวณหา Fibonacci number ลำดับที่ n เมื่อลำดับของ Fibonacci number เป็น ^{1 2 3 4 5 6} 1 1 2 3 5 8 ...
2. เขียนโปรแกรมเพื่อคำนวณหา Fibonacci number โดยไม่ใช้ recursive function เพื่อเปรียบเทียบความแตกต่าง (ในแง่ความยากง่าย และความเร็วในการทำงาน)
3. ฝึกเขียนโปรแกรมเพื่อแก้ปัญหาดังต่อไปนี้ 2-6 (ในหัวข้อความสัมพันธ์เวียนเกิด) โดยอาศัย recursive function
4. จงเขียนโปรแกรมเพื่อแก้ปัญหاتower of Hanoi (รายละเอียดอธิบายเพิ่มเติมในคาบ)
5. จงเขียนโปรแกรมเรียงลำดับข้อมูลด้วยวิธีการ merge sort โดยอาศัย recursive function
6. จงเขียนโปรแกรมเพื่อค้นหาข้อมูลในรายการข้อมูลที่เรียงลำดับแล้ว ด้วยวิธีการของ binary search โดยอาศัย recursive function



Hanoi



- Input

3 A C

11 นาที ✓



$2^2 - 1 = 3$
AB AC = 2

AB AC BC

ต้องการย้ายแผ่นไม้จำนวน 3 แผ่น จาก A ไป C

จำนวนแผ่น ต้นทาง ปลายทาง (A/B/C)

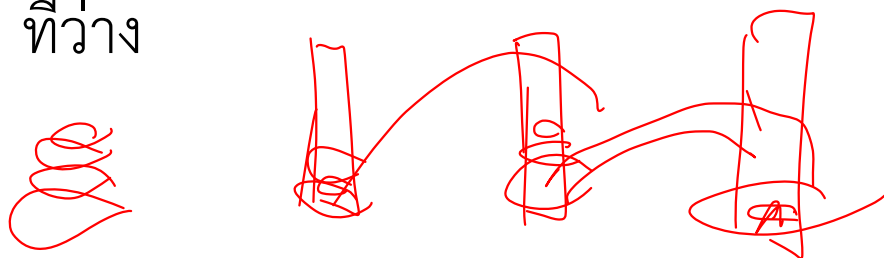
- Output

AC AB CB AC BA BC AC 7

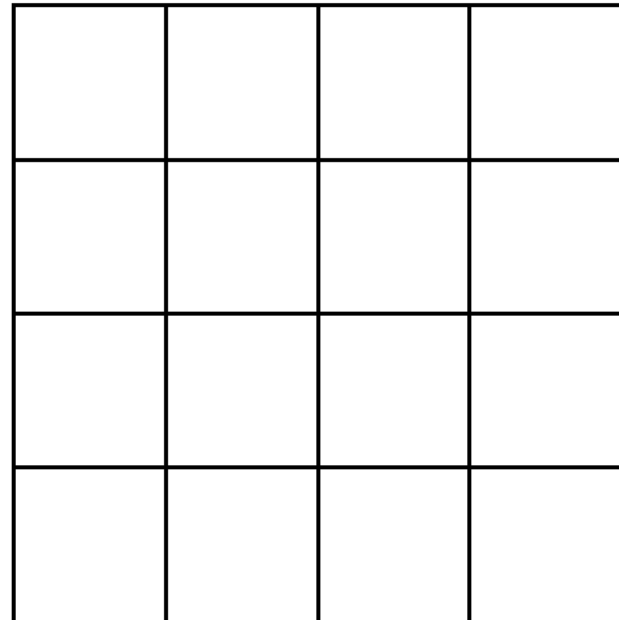
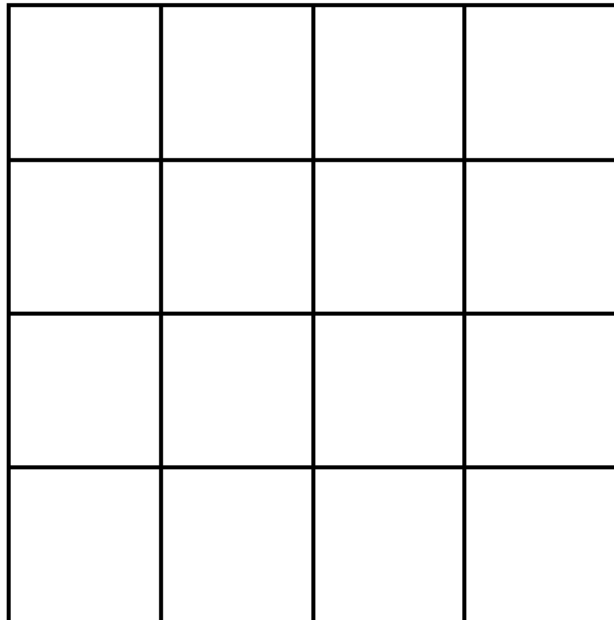
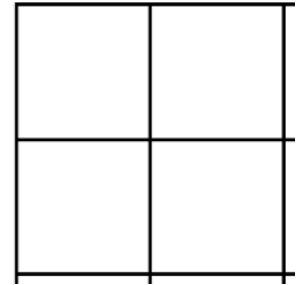
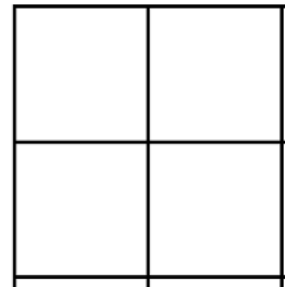
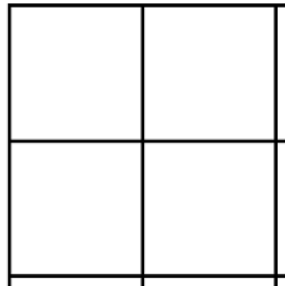
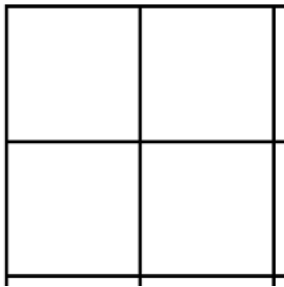
$2^n - 1$

ผลลัพธ์ คือ ลำดับการย้ายทั้งหมด และจำนวนครั้งของการย้าย

เว้นวรรค 1 ที่ว่าง



L-Shape Problem



L-Shape Problem

